The ATLAS online High Level Trigger framework: experience reusing offline software components in the ATLAS trigger

Werner Wiedenmann on behalf of the ATLAS Collaboration

Department of Physics, University of Wisconsin, Madison, Wisconsin

E-mail: Werner.Wiedenmann@cern.ch

Abstract. Event selection in the ATLAS High Level Trigger is accomplished to a large extent by reusing software components and event selection algorithms developed and tested in an offline environment. Many of these offline software modules are not specifically designed to run in a heavily multi-threaded online data flow environment. The ATLAS High Level Trigger (HLT) framework based on the GAUDI and ATLAS ATHENA frameworks, forms the interface layer, which allows the execution of the HLT selection and monitoring code within the online run control and data flow software. While such an approach provides a unified environment for trigger event selection across all of ATLAS, it also poses strict requirements on the reused software components in terms of performance, memory usage and stability. Experience of running the HLT selection software in the different environments and especially on large multi-node trigger farms has been gained in several commissioning periods using preloaded Monte Carlo events, in data taking periods with cosmic events and in a short period with proton beams from LHC. The contribution discusses the architectural aspects of the HLT framework, its performance and its software environment within the ATLAS computing, trigger and data flow projects. Emphasis is also put on the architectural implications for the software by the use of multi-core processors in the computing farms and the experiences gained with multi-threading and multi-process technologies.

1. Introduction

ATLAS [1] is one of the two large general purpose experiments at the Large Hadron Collider LHC at CERN. It covers a widely diversified physics program [2], ranging from discovery physics to precision measurements of Standard Model parameters and understanding the mechanism of electroweak symmetry breaking. Hardware construction and installation is to a large extent completed and a description of the apparatus can be found elsewhere [3]. At the design luminosity of 10^{34} cm^{-2}s^{-1} LHC will produce *pp*-collisions with a center of mass energy of $\sqrt{s} = 14$ TeV and with a bunch crossing rate of 40 MHz. About 25 overlapping interactions per bunch crossing from inelastic *pp* interactions are expected at these operational parameters. To keep interesting physics processes and to reduce the 10^9 interactions per second to an acceptable event output rate of a few hundred Hz for event storage highly selective trigger systems are required. In a first stage LHC is expected to deliver colliding beams in 2009 with a center of mass energy of $\sqrt{s} = 10$ TeV and a luminosity of 10^{31} cm^{-2}s^{-1}. Physics programs and commissioning efforts use therefore these operational parameters for startup preparations.

2. The ATLAS Trigger

Online event selection in the ATLAS trigger is done in three levels, with the hardware based Level-1 trigger and the two software based triggers, Level-2 and Event Filter (EF). Level-2 and EF together form the High Level Trigger (HLT) [4] [5]. The event selection in the HLT triggers proceeds in steps for feature extraction and hypothesis decisions. At the end of each step the step results are checked against abstract physics signatures defined in trigger menus. While the Level-2 reconstructs localized regions, the EF attempts a full offline-like event reconstruction guided by the *Level-2 Result* with more complete calibration, alignment and magnetic field data.

The Level-1 trigger [6] is implemented in custom hardware and reduces the initial event rate to about 75 kHz. It can be upgraded to 100 kHz. The Level-1 decision is based on data from the calorimeters and the muon detectors. For accepted events small localized regions in pseudo rapidity η and azimuthal angle ϕ centered on the high p_T objects identified by the Level-1 trigger are determined. Each Region of Interest (RoI) contains the type and the thresholds passed of the associated high p_T candidate objects.

The Level-2 trigger's selection process is guided by the RoI information and uses full granularity event data within a RoI from all detectors for it's decision process. In this way, only 2-4% of the full event data are transfered to the Level-2 trigger. The selection algorithms request data from the Read Out Buffers (ROB) for specific detectors in a RoI for each processing step. The data are held in the ROBs until the Level-2 trigger accepts or rejects the event. The Level-2 output rate is about 3 kHz with typical event decision times of 40 ms for a current quad-core CPU with 2 GHz clock frequency.

If an event is accepted by Level-2, the Event Builder collects all the event data fragments from the ROBs. The complete event is then made available to the EF for the final stage of trigger selection. Here, more complex algorithms provide a further rate reduction to about 200 Hz with typical event decision times of up to 4 s.

3. High Level Trigger Hardware and Software Environment

It is foreseen to install for the final HLT event selection farm 17 racks with Level-2 processors and 62 racks with EF processors. Each rack hosts 31 one unit high machines with two CPU sockets. The processors are network booted and every rack has its own local file server. Out of the 79 racks in total 28 racks will be freely configurable either for use in Level-2 or in EF. In this way processing power can be distributed according needs to Level-2 or EF. Presently the ATLAS HLT processor farm consists of 27 dual use HLT processor racks, which corresponds to about 35% of the final HLT farm size. Most of the installed processors are based on quad-core Intel Harpertown [7] CPUs with 2.5 GHz clock frequency and 2 GByte of memory per processor core.

Level-2 event selection algorithms run inside the Level-2 Processing Units [9] [10] (L2PU). Each L2PU can process events in parallel in concurrent worker-threads. Multi-threading minimizes overheads from context-switching and avoids stalling the CPU when waiting for requested RoI data from the readout system, or when publishing monitoring information. While this ansatz allows for an efficient use of multi-CPU and multi-core processor resources, it requires that all software running in the L2PU is thread-safe. The technical aspects of multi-threading are handled by the data flow software itself, including creation and deletion of threads and synchronization of resources. The EF uses more offline like HLT selection software and executes it in concurrent EF Processing Tasks (EFPT). The EFPTs are controlled by the EF data flow software, which provides also full event data via shared memory to the processing tasks.

All aspects of data movement and application control are handled by the online data flow and run control software. To a large extent these software components use multi-threading technologies to deal with asynchronous service and control requests. The ATLAS HLT framework, with the so called *HLT Steering Controller* [8], interfaces the HLT event selection

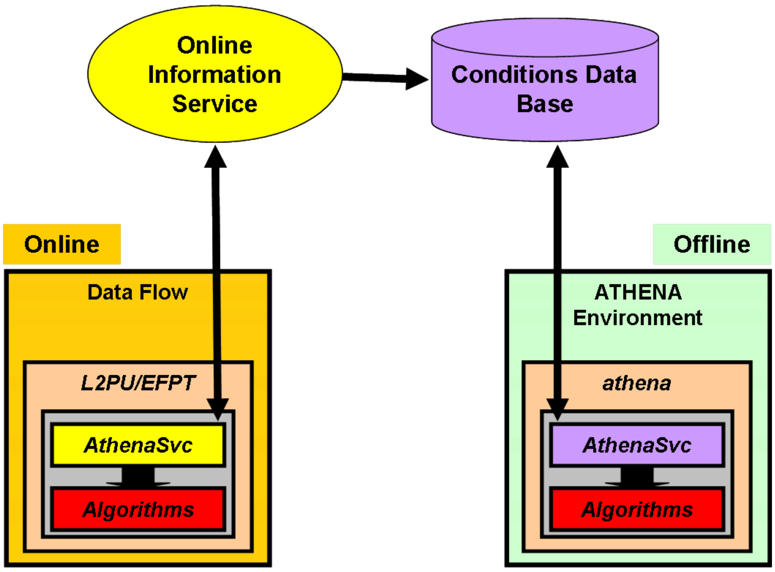


Figure 1. Schematic view of an ATHENA service which reads in online specific detector conditions from the online Information Service *IS* and presents them to the event selection algorithms in the same format as the corresponding service implementation does in offline. There the data are read directly from the offline conditions data base. In this way algorithms have a complete transparent access to their conditions data. The arrow between the online Information Service and the offline conditions data base indicates that selected data delivered by IS are also archived asynchronously to the conditions data base.

algorithms to the online run control and data flow software, which completely controls it in terms of finite state machine transitions and management of the event loop. The *HLT Steering Controller* provides for the HLT event selection software the access to the online configuration system and the access to event data, either directly from ROBs in Level-2 or from full events in EF. Further it handles error conditions arising from algorithm execution in the online data flow context. After the event selection code has finished to process the event, it packs the detailed event decision record into a raw data fragment and forwards it together with the event streaming information to the online data flow. The HLT event selection framework [11] itself forms in online a software layer inside the HLT framework and constitutes the run environment for the trigger algorithms. It is common to Level-2 and EF and is composed of four main components. The *HLT Steering* schedules the *HLT Algorithms* corresponding to the input seed, so that all necessary data for a trigger decision are produced. Information about event specific quantities is exchanged via components of the *Event Data Model* (EDM). During event processing data are stored and accessed through *Data Managers*. This allows to hide platform- and storage technology-specific details of data access from the algorithms. The *HLT Algorithms* either reconstruct new event quantities or check trigger hypotheses with previously computed event features.

Since the EF provides an offline-like, process based, environment, the HLT event selection software is naturally based on the ATLAS offline reconstruction and analysis environment ATHENA [12], which itself is build on the GAUDI [13] framework. This allows for an efficient code reuse from a large software basis with many contributors and the implementation of a “physicist-friendly” environment for trigger algorithm development. With a common code base for the online and the offline software, the HLT guarantees also the consistency of trigger performance evaluations, which are mostly performed in an offline setup. Examples of re-utilized components are the storage manager, the EDM, the detector description, the services for handling conditions data and many reconstruction tools, which are already developed by the offline community. Only the *HLT Steering* framework and certain specialized trigger algorithms remain HLT specific developments. Also the HLT framework is based to a large extent on the ATHENA architecture and re-uses whenever possible core offline services in online. Examples are services which handle the complex ATLAS detector description, the alignment and conditions data and the conversion of event data to high level EDM objects. A key feature for providing transparent access mechanisms to data in offline and online, is GAUDIs name based service architecture. It allows to implement services with different online back-ends and to configure them with the same name as in offline. Algorithms will retrieve the services according to their name and their abstract interface definition without knowing the detailed implementation. An example is the access to data fragments in the ROBs for Level-2, where a special online ROB data provider service directly contacts over the network the ATLAS readout system, retrieves the required fragments and presents them to the selection algorithms as if they would have been read from a complete raw data event like in offline. Further examples include services which read selected detector conditions data from the online information system and present them to the algorithms as if they would have been retrieved from the offline conditions database. This situation is shown schematically in figure 1. A special case forms the online job configuration service, where the complete trigger configuration is read from a database instead from Python job setup scripts, as in offline.

4. Software Development Model

A similar development model as in offline has been adopted for HLT code. Since the same interfaces are available in the EFPT and the L2PU environment the code developed in the offline environment can be directly downloaded in binary form to the HLT processors. Figure 2 shows this relation between an online data flow setup and an offline development environment for HLT software.

However testing of HLT code is more complicated since it runs inside the L2PU and EFPT data flow applications and requires therefore that developers setup data flow systems. The data flow software can be configured to run either as single or multi-node system. A single node

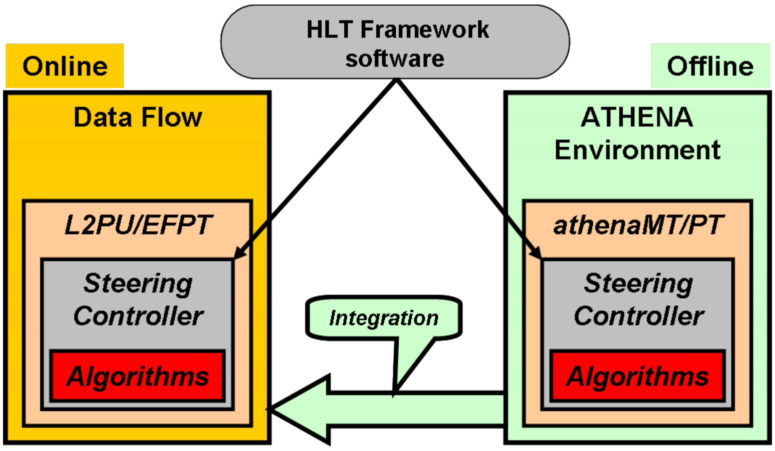


Figure 2. A typical development and prototype setup. ATHENA is the basic algorithm execution environment. It is shared by the online and the offline environments. Algorithms are developed in the offline environment and tested e.g. with the L2PU emulator *athenaMT* or the EFPT emulator *athenaPT*. The same binary libraries containing the event selection algorithms are then used in real data flow setups.

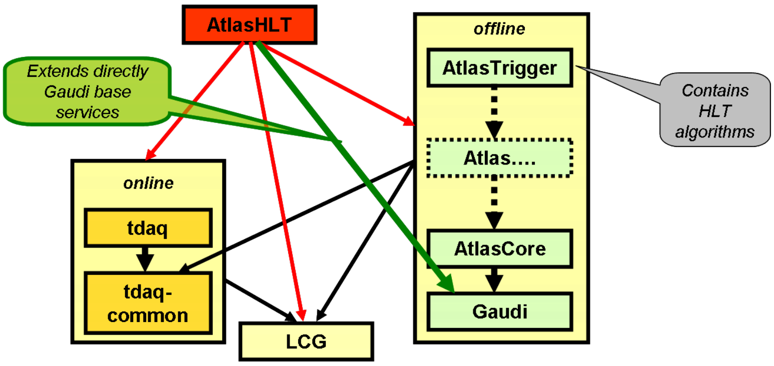


Figure 3. A schematic view of the dependency structure of the *AtlasHLT* software project within the ATLAS software projects. The *AtlasHLT* software project depends on most of the other software projects and is therefore last in the build sequence.

system starts all the required data flow and infrastructure applications in the same processing node, while a multi-node system distributes them over various nodes. The setup of a complex data flow system for application testing is in both cases a nontrivial task and requires also access to the necessary hardware resources.

Two command line applications, *athenaMT* [14] and *athenaPT* were therefore created, which emulate internally the run environment of a L2PU or of a EFPT. They are written in Python and share to a large extent a common code basis. Differences arise only in the way the HLT event selection software accesses raw data and in the emulation of the multi-threaded event selection in Level-2. To simulate the raw data access via network requests for Level-2, *athenaMT* loads the raw data fragments for each event into memory and provides them on algorithm request to the event selection software. In a similar way *athenaPT* provides the full raw event to the HLT code. Both emulators allow in an interactive mode to cycle through the ATLAS trigger finite state machine for testing if newly created code is compatible with trigger operation. Command line options and debug aids for the emulators have been implemented as close as possible to the ones available with the offline ATHENA run script. In this respect developers with an "offline background" can more easily get acquainted with HLT development and they need not to be familiar with detailed technical aspects of the data flow software. Further in their development process they are also shielded from changes in the data flow part of the software. They can concentrate exclusively on the HLT software and are able to perform a large variety of tests with these command line applications. It is clear, however, that the final certification of the HLT software has to be done on a large distributed system.

5. The AtlasHLT Project

The packages which belong to the HLT framework form the so called *AtlasHLT* software project. It is structured like other offline software projects [15] and uses the same version control, build and testing tools. Due to its nature as an interface between data flow and offline, it is the only ATLAS software project which can directly depend on data flow software and on all other ATLAS software projects (see also figure 3). For this reason the *AtlasHLT* software project is build as the last software project after all other dependencies are ready. It is clear that due its dependency structure the releases of the *AtlasHLT* project have to be coordinated with trigger and data acquisition (TDAQ) releases and with offline releases. However other dependencies between these two projects already exist due to the common use of *LCG* software and the common software project *tdaq-common*, which mainly hosts the code for the ATLAS raw event data format.

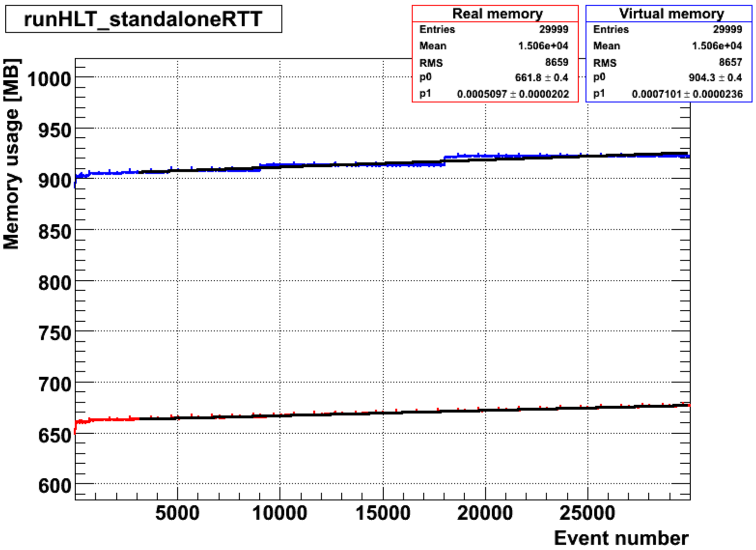


Figure 4. Example of an automatic memory leak test for Level-2 code run every night after the *AtlasHLT* developer builds. The graph shows the memory usage for the tested framework and selection code as a function of the number of processed events. The upper curve shows the evolution of the virtual memory and the lower curve the evolution of the real memory. For Level-2 performance targets require memory leaks to be smaller than about 10 Bytes/event. The exact location of a leak has to be localized by developers with other tools.

Developers administrate their software components with the ATLAS *Tag Collector* and every night a new test release is build. Also every night automatic code tests are launched using mainly the emulator applications *athenaMT* and *athenaPT*. Simple single node data flow setups complement the test suite with automatic code tests in data flow partitions. Figure 4 shows an example of a memory leak test, performed with the release test suite. Since the reused offline code modules are executed in trigger orders of magnitude more often than on the offline reconstruction farms, testing of code robustness and careful control of memory leaks is of great importance. E.g. performance targets for trigger require memory leaks in Level-2 to be smaller than 10 Bytes/event and to be less than 1 kByte/event in EF. In close collaboration with offline developers performance monitoring tools have been developed which help to follow the execution time and memory evolution of the code over the different release cycles in the offline and in the *AtlasHLT* projects.

6. Operational Experience

The HLT framework has been used now for over more than five years. It was first deployed in a setup with detector components for the ATLAS test beam in 2004 [16], it was used for many data taking periods with cosmics in 2007 and 2008, including the combined ATLAS cosmic runs, and it was configured in pass through mode for first LHC events in September 2008. These data taking periods exercised all core software components from algorithm configuration with the trigger data base to event streaming initiated by algorithm decisions. E.g. for the cosmic data taking period starting September 13, 2008, 216 million events were handled by the HLT framework with a data volume of 453 TByte in 400000 files (see figure 5). About 3% of the events which caused data taking problems, mainly due to corrupted raw data fragments or event time outs, were written to the debug stream. Events from this stream were re-analyzed by re-running the trigger event selection code offline and they were eventually re-injected into

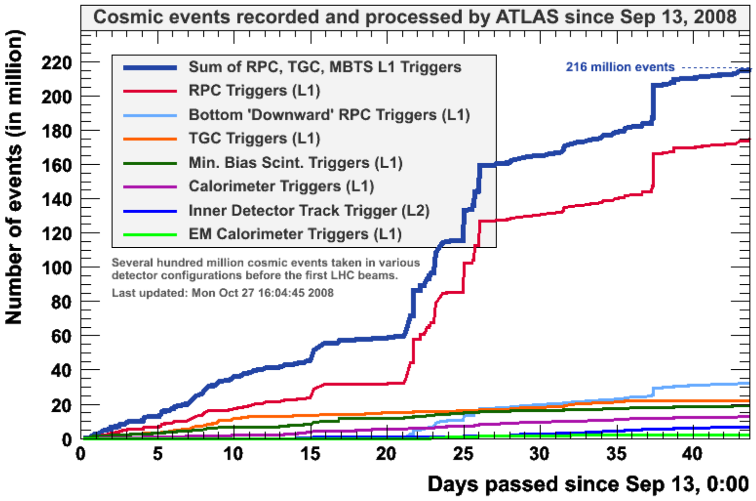


Figure 5. Number of cosmic events recorded with different triggers in the ATLAS cosmic run starting September 13, 2008.

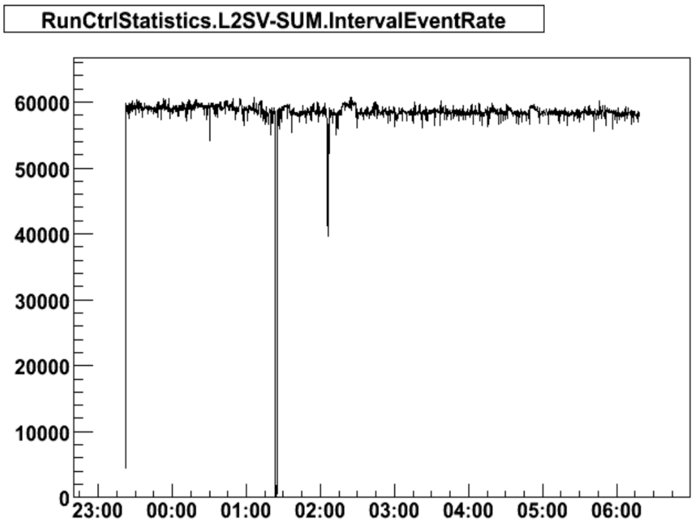


Figure 6. Input rate in Hz (y-axis) to the Level-2 trigger in a test run lasting seven hours. Simulated data were preloaded into the read out system and pushed through a setup with 2880 Level-2 Processing Units, which corresponds to about 70% of the final Level-2 system size. A rate of 60 kHz could be sustained. The sharp drops in rate are caused by system cron jobs.

the analysis sample. For reprocessing these events and debugging the various problems the emulators *athenaMT/PT* were used.

While cosmic data taking provides an invaluable operational experience with an integrated detector setup, the TDAQ system and the HLT framework are not pushed to its performance and throughput limits. These limits were explored in several so called “TDAQ Technical Runs”. For these periods the read out system was preloaded with simulated data. The data fragments from different sub detectors were assigned in a realistic way to the read out drivers and the full HLT event selection menu for a luminosity of 10^{31} cm^{-2}s^{-1} was configured. All available HLT processors were attributed to the Level-2 farm running 2880 L2PU instances, which corresponds

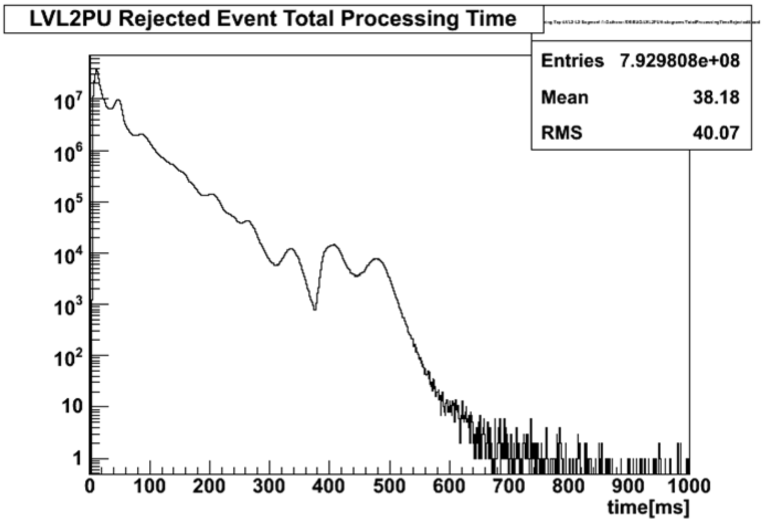
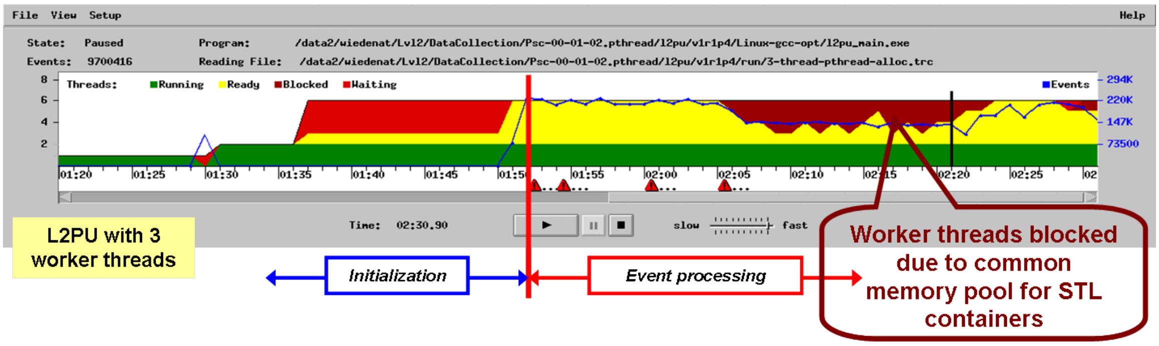


Figure 7. Processing time for rejected events in Level-2. These events will constitute more than 95 % of the events handled by Level-2. The timing is well within the expected processing time of about 40 ms for the used CPU type.



State: Paused
Events: 9700416
Program: /data2/wiedenat/Lvl2/DataCollection/Psc-00-01-02.pthread/l2pu/v1r1p4/Linux-gcc-opt/l2pu_main.exe
Reading File: /data2/wiedenat/Lvl2/DataCollection/Psc-00-01-02.pthread/l2pu/v1r1p4/run/3-thread-pthread-alloc.trc
Time: 02:30.90

Figure 8. Event processing in three concurrent worker threads. It can be seen that for a substantial fraction of time the worker threads block each other during event processing. This is due to an inefficient memory allocation scheme used in an older version of the *Standard Template Library*.

to about 70 % of the final system. In a seven hour lasting run, an input rate of 60 kHz could be sustained (figure 6). This value corresponds to about 80 % of the design value. With less than 40 ms processing time (figure 7) for rejected events in Level-2, the HLT code performs within the expected limits on the used HLT processors.

7. Event Parallelism and Multi-Core Processors

The original ATLAS design for the HLT event selection applications exploits event parallelism for reaching the required event throughput. In Level-2 concurrent worker threads execute the selection code on each processor in parallel, while concurrent processing tasks work in parallel on the full events provided to the EF. A typical HLT processor consisted of a dual CPU machine with a single core CPU per socket. Performance for these CPUs would increase with increasing clock frequency. It was foreseen to start on each Level-2 processor one instance of a L2PU application with up to three worker threads and two instances of EFPTs per EF processor.

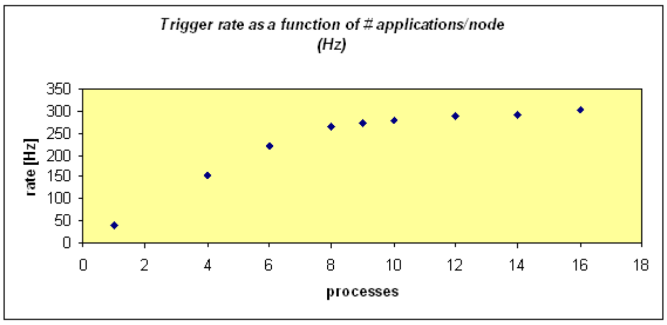


Figure 9. Scaling of the event throughput rate with the number of HLT selection process instances on a dual CPU quad-core machine. There is almost linear scaling up to 8 process instances, i.e. one process instance per CPU core. Running more selection processes than available CPU cores does not improve the throughput anymore.

It became already very early obvious that it is very difficult to maintain thread safe HLT event selection code in an open developer community. The availability of the same algorithm interfaces in HLT allowed a very fast development and deployment of new selection chains by reusing many components already developed in offline. These components were not always designed to run in a multi-threaded environment and a redesign had often far reaching consequences for key core components in offline. While in the beginning certain selection chains could run multi-threaded, it proved to be very difficult to maintain this capability over release cycles. Another mayor problem arose from the basic system libraries themselves. During tests it was observed that the event throughput in a L2PU didn't scale in the expected way with the number of worker-threads. This was due to the use of a common memory pool for container objects in the default memory allocation scheme of the *Standard Template Library* (STL). resulting in frequent mutual blocking of the worker threads as figure 8 shows. The event processing model of Level-2 favors a scheme where every thread allocates its own memory pool. Such an allocation scheme was available in the STL and after optimizing the code with it, the expected scaling behavior was observed. However, external utility libraries had to be also compiled with this allocation scheme and were not always available in this form. This made it very difficult to run even thread safe code in an efficient way. As a consequence the L2PU is now configured to run with one HLT event selection worker thread, but it still uses all the multi-threaded infrastructure for data retrieval, monitoring and run control. The required throughput is achieved by starting one L2PU instance per CPU core. With the end of the "frequency scaling area" and the introduction of multi-core CPUs with lower clock frequency the number of applications per HLT processor is multiplied by the total number of available CPU cores, since one L2PU and one EFPT application is launched per core. While an almost linear increase of the event throughput with the number of HLT applications is observed (figure 9), the required resources, like available system memory and number of network connections, increase also. The expected launch of many-core processors in the coming years and the possible use of less machines with more concentrated compute power to run the HLT selection applications at the same decision rate as now, would also require more available bandwidth for data input to the processors. On the other hand, by running the different selection levels in one machine, the input bandwidth requirements could be relaxed due to the increased time spend on an event. Both scenarios will however require further enhancements to the framework to optimize the use of multi- and many-core systems (see also Refs. [17], [18]).

8. Conclusions

The presented implementation of the HLT framework enables the reuse of offline software components throughout the ATLAS High Level Triggers. It realizes a homogeneous software and development environment from the Level-2 trigger to offline and allows to benefit from a big developer community and software base. Close collaboration with the offline developers made it possible to reach the execution speed, robustness and memory leak targets necessary for trigger operation. The HLT framework with the event selection algorithms has now been successfully used in many data taking periods and provides the expected throughput. The ability to develop, test and optimize trigger algorithms in offline is successfully used to establish the ATLAS HLT selection menu. The recent changes in processor technology and the move to multi- and many-core processors requires however further framework developments to support optimally these hardware platforms.

References

[1] ATLAS Collaboration, *ATLAS: Technical proposal for a general-purpose pp experiment at the Large Hadron Collider at CERN*, CERN/LHCC/94-43 (1994)

[2] ATLAS Collaboration, *ATLAS Detector and Physics Performance Technical Design Report*, CERN/LHCC/99-014 and 99-015 (1999)

[3] The ATLAS Collaboration, Aad G et al, *The ATLAS Experiment at the CERN Large Hadron Collider*, JINST 3 (2008) S08005

[4] ATLAS Collaboration, *ATLAS High-Level Triggers, DAQ and DCS Technical Proposal*, CERN/LHCC/2000-017 (2000)

[5] ATLAS Collaboration, *ATLAS High-Level Trigger Data Acquisition and Controls, Technical Design Report*, CERN/LHCC/2003-022 (2003)

[6] ATLAS Collaboration, *ATLAS Level-1 Trigger Technical Design Report*, CERN/LHCC/98-014 (1998)

[7] Wikipedia, http://en.wikipedia.org/wiki/Xeon

[8] Armstrong S et al, *Studies for a Common Selection Software Environment in ATLAS: From Level-2 Trigger to the Offline Reconstruction*, *IEEE Trans. Nucl. Sci.*, **51** (2004) 915-920

[9] Bogaerts A and Wickens F, *LVL2 Processing Unit Application Design*, EDMS Note, ATL-DH-ES-0009 (2001)

[10] Armstrong S et al, *The Second Level Trigger of the ATLAS Experiment at CERN's LHC*, *IEEE Trans. Nucl. Sci.*, **51** (2004) 909-914.

[11] PESA Software Group (editor M. Elsing), *Analysis and Conceptual Design of the HLT Selection Software*, ATLAS Internal Note, ATL-DAQ-2002-013 (2002)

[12] ATLAS Collaboration, *ATLAS Computing Technical Design Report*, ATLAS Internal Note, ATLAS TDR–017 and CERN-LHCC-2005-022, (2005)

[13] The Gaudi Project. Available: http://proj-gaudi.web.cern.ch/proj-gaudi/

[14] Anjos A, Pinto P and Wiedenmann W, *athenaMT and Level-2 Software Integration*, EDMS Note, ATL-DH-EN-0009 (2005)

[15] Obreshkov E et al, *Organization and Management of ATLAS Offline Software Releases*, *Nucl. Instr. and Methods in Phys. Res. Sect. A*, Vol **584**, Issue 1 (2008), 244-251

[16] Anjos A et al, *Deployment of the ATLAS High-Level Trigger*, *IEEE Trans. Nucl. Sci.*, **53** (2006) 2144-2149

[17] Binet S et al, *Harnessing multicores:strategies and implementations in ATLAS*, CHEP 2009, contribution 244, Prague, March 21-27 (2009)

[18] Innocente V, *The Challenge of Adapting HEP Physics Software Applications to Run on Many-Core CPUs*, CHEP 2009, contribution 520, Prague, March 21-27 (2009)